TP4 – Classification bayésienne

Réalisé par : Zahra BENSLIMANE

Chargé de TP : Flavien Lebrun

Chargée de cours : Catherine Achard

Dans ce TP,

I. Chargement et visualisation des données

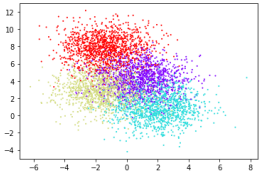
Charger les données (TP5a.npy) et visualiser les

Questions

Combien y a-t-il de points dans la base d'apprentissage ? Dans la base de test ? Quelle est la dimension des données ?

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
```

```
[X_train, y_train, X_test, y_test] = np.load("TP5a.npy",allow_pickle=True)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=1, cmap='rainbow');
plt.show()
```



```
print("X_train.shape = ", X_train.shape)
print("X_test.shape = ", X_test.shape)

print("\nIl y a {} points dans la base d'apprentissage.".format(X_train.shape[0]))
print("Il y a {} points dans la base de test.".format(X_test.shape[0]))
```



```
print('\nLes données sont de dimension : {}'.format(X_train.shape[1]))
```

```
X_train.shape =  (4900, 2)
X_test.shape =  (100, 2)

Il y a 4900 points dans la base d'apprentissage.
Il y a 100 points dans la base de test.

Les données sont de dimension : 2
```

II. Arbre de décision

a. Principe des arbres de décision

On réalise la classification avec un arbre de décision (fonction en annexe visualize_classifier):

--> Approche discriminative : Détermine directement $p(y|\mathbf{x})$

Idée

• Classer avec un ensemble de règles.

• Une suite de décisions permet de partitionner l'espace en régions homogènes

• La difficulté consiste à créer l'arbre à partir de la base d'exemple étiquetée

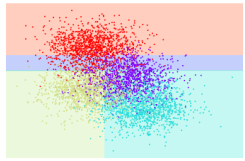
```
def visualize_classifier(model, X, y):
 ax = plt.gca()

 # Plot the training points
 ax.scatter(X[:, 0], X[:, 1], c=y, s=1, cmap='rainbow',
 clim=(y.min(), y.max()), zorder=3)
 ax.axis('tight')
 ax.axis('off')
 xlim = ax.get_xlim()
 ylim = ax.get_ylim()
 xx, yy = np.meshgrid(np.linspace(*xlim, num=200),np.linspace(*ylim, num=200))
 Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

 # Create a color plot with the results
 n_classes = len(np.unique(y))
 contours = ax.contourf(xx, yy, Z, alpha=0.3,
 levels=np.arange(n_classes + 1) - 0.5,cmap='rainbow', zorder=1)
 ax.set(xlim=xlim, ylim=ylim)
 plt.show()
```

```
tree1 = DecisionTreeClassifier(criterion='entropy', max_depth = 2)
tree1.fit(X_train, y_train)
visualize_classifier(tree1, X_train, y_train)

tree.plot_tree(tree1)
text_representation = tree.export_text(tree1)
print(text_representation)
```

```
|--- feature_1 <= 5.19
|   |--- feature_0 <= -0.56
|   |   |--- class: 2
|   |--- feature_0 >  -0.56
|   |   |--- class: 1
|--- feature_1 >  5.19
|   |--- feature_1 <= 6.91
|   |   |--- class: 0
|   |--- feature_1 >  6.91
|   |   |--- class: 3
```

Questions Que représente la variable max_depth ?

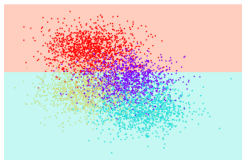
Quelles règles de décisions ont été prises ?

Faire varier max_depth de 1 à 20 (en supprimant les 3 dernières lignes) et commenter les résultats obtenus visuellement. Retrouvez-vous toutes les découpes ?

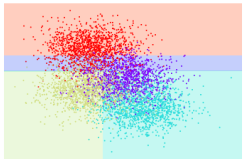
```
max_depth_Tab = np.arange(1,20)

for max_depth in max_depth_Tab:
  tree1 = DecisionTreeClassifier(criterion='entropy', max_depth = max_depth)
  tree1.fit(X_train, y_train)
  print("                max_depth : ", max_depth)
  visualize_classifier(tree1, X_train, y_train)
```

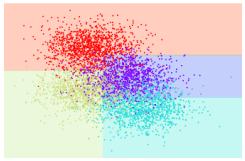

max_depth : 1



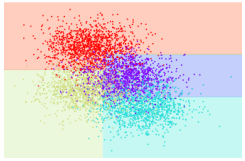
max_depth : 2



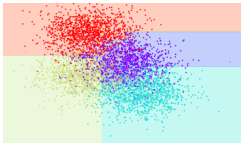
max_depth : 3



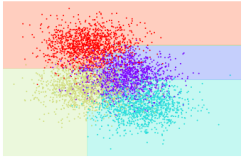
max_depth : 4



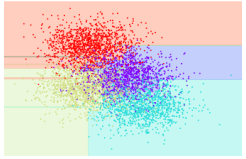
max_depth : 5



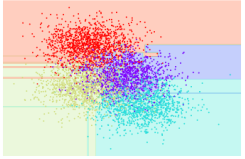
max_depth : 6



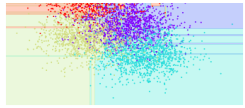
max_depth : 7



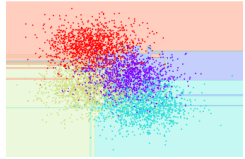
max_depth : 8



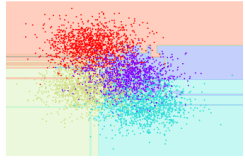
max_depth : 9



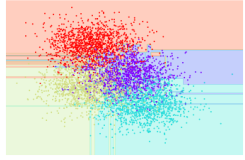
max_depth : 10



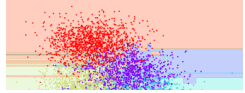
max_depth : 11

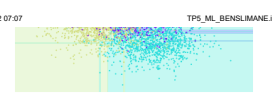


max_depth : 12

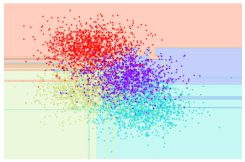


max_depth : 13

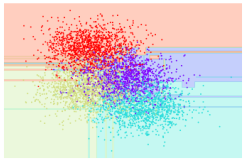




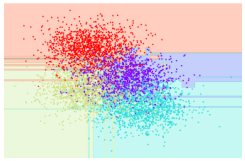
max_depth : 14



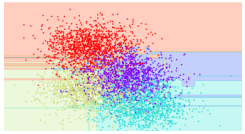
max_depth : 15



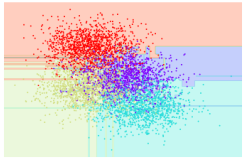
max_depth : 16



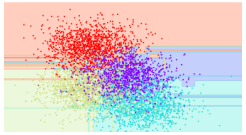
max_depth : 17



max_depth : 18



max_depth : 19



b. Performance d'un classifieur muti-classes

Choisissez la variable max_depth qui vous parait visuellement la plus appropriée et réaliser la classification sur la base de test avec :

max_depth optimal = 6

```
tree = DecisionTreeClassifier(criterion='entropy', max_depth = 6)
tree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=6)
```

```
y_pred = tree.predict(X_test)
C= confusion_matrix(y_test, y_pred)
print(C)
print(classification_report(y_test, y_pred))
print('Accuracy= ',accuracy_score(y_test, y_pred))
```

```
[[16  2  5  3]
 [ 3 24  0  0]
 [ 5  3 12  1]
 [ 3  0  0 23]]
              precision    recall  f1-score   support

           0       0.59      0.62      0.60        26
           1       0.83      0.89      0.86        27
```



```
           2       0.71      0.57      0.63        21
           3       0.85      0.88      0.87        26

    accuracy                           0.75       100
   macro avg       0.74      0.74      0.74       100
weighted avg       0.75      0.75      0.75       100

Accuracy=  0.75
```

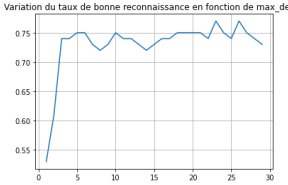
c. Optimisation de la profondeur de l'arbre

Faire varier max_depth et estimer pour chaque valeur le taux de reconnaissance. Conclusion sur l'évolution du taux de reconnaissance en fonction de la profondeur de l'arbre. Afficher le partitionnement de l'espace obtenu pour l'arbre avec le meilleur taux de classification

```
accuracy_score_Tab = []
max_depth_Tab = np.arange(1,30)

for max_depth in max_depth_Tab:
  tree = DecisionTreeClassifier(criterion='entropy', max_depth = max_depth)
  tree.fit(X_train, y_train)
  y_pred = tree.predict(X_test)
  C= confusion_matrix(y_test, y_pred)
  accuracy_score_Tab.append(accuracy_score(y_test, y_pred))

plt.figure(); plt.plot(max_depth_Tab,accuracy_score_Tab); plt.title("Variation du taux de
plt.grid(); plt.show()
print("argmax = ",np.argmax(accuracy_score_Tab)+1)
```




```
argmax =  23
```

III. Arbre de décision sur des données de grande dimension

a. Classification avec un arbre de décision Charger maintenant les données « TP5b.npy »

```
[X_train, y_train, X_test, y_test] = np.load("TP5b.npy",allow_pickle=True)
```


Quelle est leur dimension ? Combien y a t-il de points en apprentissage et en test ? Combien y a-t-il de classes ?

```
print("X_train.shape = ", X_train.shape)
print("X_test.shape = ", X_test.shape)

print("\nIl y a {} points dans la base d'apprentissage.".format(X_train.shape[0]))
print("Il y a {} points dans la base de test.".format(X_test.shape[0]))

print('\nLes données sont de dimension : {}'.format(X_train.shape[1]))
print("Les classes = ",np.unique(y_train))
```

```
X_train.shape =  (966, 50)
X_test.shape =  (322, 50)

Il y a 966 points dans la base d'apprentissage.
Il y a 322 points dans la base de test.

Les données sont de dimension : 50
Les classes =  [0 1 2 3 4 5 6]
```

dimensions 50 peut pas afficher

b. Forêt d'arbres aléatoires

Utiliser RandomForestClassifier pour construire une forêt d'arbres aléatoires

```
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier(criterion='entropy', n_estimators=30, random_state=1)
RF.fit(X_train, y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=30, random_state=1)
```

Tester le classifieur avec les paramètres par défaut et 30 arbres. Conclusion. Utiliser la fonction GridSearchCV() pour optimiser les paramètres de la forêt

```
RF = RandomForestClassifier(criterion='entropy', n_estimators=30, random_state=1)
RF.fit(X_train, y_train)

y_pred = RF.predict(X_test)
C = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))
print('Accuracy= ',accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.31      0.47        13
           1       0.69      0.78      0.73        60
           2       0.67      0.44      0.53        27
           3       0.67      0.92      0.78       146
           4       0.40      0.16      0.23        25
```



```
           5       0.83      0.33      0.48        15
           6       0.64      0.25      0.36        36

    accuracy                           0.67       322
   macro avg       0.70      0.46      0.51       322
weighted avg       0.67      0.67      0.63       322

Accuracy=  0.6708074534161491
```

Grid Search with Cross Validation Random search allowed us to narrow down the range for each hyperparameter. Now that we know where to concentrate our search, we can explicitly specify every combination of settings to try. We do this with GridSearchCV, a method that, instead of sampling randomly from a distribution, evaluates all combinations we define. To use Grid Search, we make another grid based on the best values provided by random search

Cross Validation The technique of cross validation (CV) is best explained by example using the most common method, K-Fold CV. When we approach a machine learning problem, we make sure to split our data into a training and a testing set. In K-Fold CV, we further split our training set into K number of subsets, called folds. We then iteratively fit the model K times, each time training the data on K-1 of the folds and evaluating on the Kth fold (called the validation data). As an example, consider fitting a model with K = 5. The first iteration we train on the first four folds and evaluate on the fifth. The second time we train on the first, second, third, and fifth fold and evaluate on the fourth. We repeat this procedure 3 more times, each time evaluating on a different fold. At the very end of training, we average the performance on each of the folds to come up with final validation metrics for the model.

- n_estimators = number of trees in the foreset
- max_features = max number of features considered for splitting a node
- max_depth = max number of levels in each decision tree
- min_samples_split = min number of data points placed in a node before the node is split
- min_samples_leaf = min number of data points allowed in a leaf node
- bootstrap = method for sampling data points (with or without replacement)

```
from sklearn.model_selection import GridSearchCV

# Create the parameter grid based on the results of random search
param_grid = {
    'criterion' :  ["gini", "entropy"],
    #'bootstrap': [True,False],
    'max_depth': [5, 6,7],
    #'max_features': [2, 3],
    #'min_samples_leaf': [1, 2],
    #'min_samples_split': [2, 3],
    'n_estimators': [25, 30, 35,]
}
# Create a based model
```



```
rf = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 6)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

grid_search.best_params_
```

```
{'criterion': 'entropy', 'max_depth': 7, 'n_estimators': 30}
```

```
def evaluate(model, test_features, test_labels):
    print(test_labels.shape)
    print(test_labels)
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    mape = 100 * np.mean(errors / test_labels)
    print(mape)
    accuracy = 100 - mape
    print('Model Performance')
    print('Average Error: {:0.4f} degrees.'.format(np.mean(errors)))
    print('Accuracy = {:0.2f}%.'.format(accuracy))
    return accuracy

best_grid = grid_search.best_estimator_
#grid_accuracy = evaluate(best_grid, X_test, y_test)

best_grid.fit(X_train, y_train)

y_pred = best_grid.predict(X_test)
C = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))
print('Accuracy= ',accuracy_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.50      0.15      0.24        13
           1       0.73      0.63      0.68        60
           2       0.82      0.33      0.47        27
           3       0.60      0.96      0.74       146
           4       0.40      0.08      0.13        25
           5       1.00      0.20      0.33        15
           6       0.54      0.19      0.29        36

    accuracy                           0.62       322
   macro avg       0.66      0.36      0.41       322
weighted avg       0.63      0.62      0.57       322

Accuracy=  0.6242236024844721
```